2000

Using granules to find association rules

Eric Wah Louie
San Jose State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI®

USING GRANULES TO FIND ASSOCIATION RULES

A Thesis

Presented to

The Faculty of the Department of Mathematics and Computer Science

San Jose State University

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

by

Eric Wah Louie

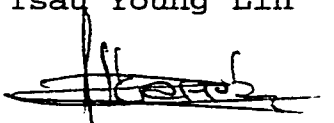
December 2000

UMI Number: 1402523

Bell & Howell Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

APPROVED FOR THE DEPARTMENT OF
MATHEMATICS AND COMPUTER SCIENCE

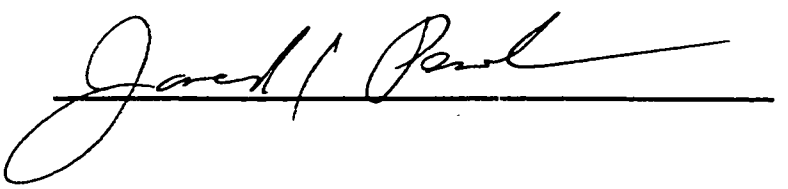
Dr. Tsau Young Lin



Dr. Mario Albarran

Dr. Porter Wong, IEEE

APPROVED FOR THE UNIVERSITY



ABSTRACT

USING GRANULES TO FIND ASSOCIATION RULES

by Eric Wah Louie

Discovering the relationships of attribute values in stored data is an essential process of assessing the semantics of the data. Typically, the process to determine the relationships compares the attribute values in the data. A granule is a compact list of tuple names having the same attribute value, and is represented in binary form. With the data converted to granules, the relationships are determined by the intersection of the granules. The significance is the speed in computation. This thesis paper explains how the representation of granules are applicable to machine architectures and how it can be used to find association rules.

To Mei, Lisa, and Leighton

Table of Contents

1. Introduction

Data mining is an exploration of the unknown semantics from stored data. The semantics of data are important pieces of information for the scientific, medical, and business communities. Meteorologists look for patterns in the data to forecast the weather conditions. Physicians, to understand the effects of certain drugs, examine the symptoms from past patients with various medications. Advertisers evaluate peer and group discussions to trace market trends. Generally, these groups search for the semantics of data to make decisions.

One area in data mining is the discovery of association rules in the data. Rakesh Agrawal introduced his method, Apriori, in the early 1990s to find association rules. The significance of his method is the use of previous knowledge to find better knowledge in the data. The impact is the reduction on the number of patterns to determine association rules. Thus, the overall execution time in finding association rules is reduced. Furthermore, his method has ignited many interests in data mining over the second half of the 1990s.

Discovering the semantics of the data is a computational task, requiring many reads of the data and many comparisons of attribute values within the data. In addition, the organization of the data is known to have a

huge affect on how quickly an answer is produced. The data model that uses granules to represent the attribute values offers a better approach to find the semantics from the data. In review, a granule is a collection of the 'names' or 'ids' in the data. This paper is based on this data model.

The major significance of this data model is the computational speed. This thesis paper explains how this data model is represented and how it used to discover association rules from the data. Lastly, this model is compared with the Apriori method of finding association rules.

2. Granule Representation

Let **U** be the universe of discourse, a classical set. The universe contains entities, and each entity in the universe has properties. These properties are referred to as elementary concepts throughout this paper. This universe is the knowledge representation of real world entities. With some conviction, the universe is the relation in relational database theory. Each entity in the universe represents one tuple in the relation, and each tuple represents one entity in the universe. These elementary concepts are the attribute values in the relation. This view is used to represent knowledge.

The universe consists of a collection of elementary concepts, named **C**. The entities in the universe are mapped to the collection of elementary concepts. In mathematical form, **M:U→C**. Each elementary concept induces a partition on U. The partition is a subset of entities in the universe. Basically, the entities with the same properties are grouped together, thus partitioning the universe. Each of these partitions is a mutually disjoint equivalence class (Lin 113). Since the universe has N elementary concepts, there are N equivalence classes. The equivalence classes are elements in the quotient set, **Q = U/M**, as well as the canonical names of the elements in quotient set.

The equivalence relations are the granules of the relation. Each unique attribute value in the relation is a granule, and each granule is a list of tuples that have the same attribute value. Only the tuple name, or the reference to the tuple, is stored in the granules.

Other than a list, the granules can be represented in bit streams. Each tuple in the relation has one unique offset position in the bit stream. The bits are in the state, SET, for those tuples having the attribute value of the granule, and the bits are in the state, CLEAR, for those tuples not having the attribute value of the granule (Chan 355). This is the bit representation of the granules of the list.

2.1. Example of Granule Representation

These concepts are applied below to a simple relation, vehicles. The relation consists of two columns: vehicle ID and vehicle type. The values in the first column identify the particular vehicles in the relation and are unique among the tuples in the relation. In this example, the values in this attribute are used to reference the tuples in the relation even though the relation has its own intrinsic way to identify tuples in the relation. The second column, vehicle type is not unique and this attribute specifies a certain property of a vehicle. The relation contains the following types of vehicles: five sedans, three mini vans, and four SUVs.

Vehicle ID	*Vehicle Type*
ID1	Sedan
ID2	SUV
ID3	Sedan
ID4	Mini Van
ID5	Mini Van
ID6	Mini Van
ID7	SUV
ID8	Sedan
ID9	Sedan
ID10	SUV
ID11	SUV
ID12	Sedan

Table 1. Vehicle relation

The quotient set for this attribute is {Sedan, SUV, Mini Van}. Each element in the quotient set represents a name to a granule and is the equivalence class that partitions the vehicle relation. The granules are shown below in two

forms: the list form and the binary form. The list form holds the the values from vehicle id. The list form has its advantages to be readable and identifiable of the tuples in the relation. The binary form of the granules contains a bit stream of zeros and ones with the offset positions marking the tuples with a particular attribute value. In either form, they represent the granules in the relation.

Vehicle Type Quotient Set in List Form
SUV = {ID2, ID7, ID10, ID11}
Mini Van = {ID4, ID5, ID6}
Sedan = {ID1, ID3, ID8, ID9, ID12}

Vehicle Type Quotient Set in Binary Form
SUV = {010000100110}
Mini Van = {000111000000}
Sedan = {101000011001}

Figure 1. Quotient Set in List and Binary Forms

Converting the vehicle relation to binary form can be done in four steps. Step 1 takes each element in the quotient set and creates a new attribute in a new table. Table 2 demonstrates step 1. Step 2 marks the appropriate attribute in the new table for each tuple's attribute value in the old table. The value '1' is placed on the attribute in the new table that is the matching attribute value of that tuple in the old table. The value '0' is placed on other attributes that do not match the attribute value of that tuple from the old table. Table 3 shows step 2. Step 3 rotates 90° the table 3 to make table 4. Finally, step 4 takes all the values in the row and compresses them into a

binary stream of zeros and ones. Table 5 displays the names and the binary list representations of each granule.

ID	Sedan	Mini Van	SUV

Table 2: Step one

ID	Sedan	Mini Van	SUV
1	1	0	0
2	0	0	1
3	1	0	0
4	0	1	0
5	0	1	0
6	0	1	0
7	0	0	1
8	1	0	0
9	1	0	0
10	0	0	1
11	0	0	1
12	1	0	0

Table 3: Step two

	1	2	3	4	5	6	7	8	9	10	11	12
SUV	0	1	0	0	0	0	1	0	0	1	1	0
Mini Van	0	0	0	1	1	1	0	0	0	0	0	0
Sedan	1	0	1	0	0	0	0	1	1	0	0	1

Table 4: Step Three (The 90° Rotation)

Quotient Set	Binary Form
SUV	{010000100110}
Mini Van	{000111000000}
Sedan	{101000011001}

Table 5: Step Four (The Compaction)

The three binary representations are the canonical names of each granule. For a single column relation, the union of the three binary representations is the relation, and the intersection of the three binary representations is empty. Therefore, the granules are disjoint subsets of the

relation. So now, these granules are used to find the relationships between attribute values. This is granular computing, and the next section explains how this is applicable to the discovery of association rules.

3. Using Granules to Find Association Rules.

A common method to summarize the data is to find the association rules or patterns (Agrawal 207). The task to find association rules may be difficult because the data may contain inaccuracies and inconsistencies. Or, the data may be extremely large. Furthermore, the number of patterns may be incomprehensible. So finding association rules must be flexible in declaring when a pattern is an association rule.

Formally, an association rule exists in the given relation if a certain percentage of the data has that pattern. In terms of granules, a combination of granules is postulated as a pattern. The pattern becomes a rule if the intersection of the granules results in a count that exceeds the minimal count, 'c', of that relation. In the language of SQL, the pattern consists of attribute values that specify the predicates to the SELECT statement. On the execution of the SELECT statement, a counter is incremented for all tuples that satisfy all the predicates. If the counter exceeds the minimal percentage of the relation, the pattern is declared an association rule.

For years, many papers discussed various methods to find association rules. Many methods relied on minimizing the number of patterns to search the data. Each pattern must have the potential of being declared an association rule; otherwise, the patterns expend computational resources. This method is Rakesh's algorithm. Basically, Rakesh's idea is to generate patterns only from previous knowledge and to check that the generated patterns satisfy all previous knowledge (Agrawal 501). The same idea is used here.

The patterns are referred as combinations or candidates. The basic algorithm to find association rules in the data is the following:

1) Generate possible candidates (combinations) of attribute values.
2) Count the number of times each possible candidate appears in the relation.
3) Declare association rules of those candidates that meet or exceed the 'c' rows in the relation.
4) If more candidates can be generated from step 3, then repeat steps 1, 2, and 3 until no more candidates can be generated.

The flow chart of this cycle is pictured below.

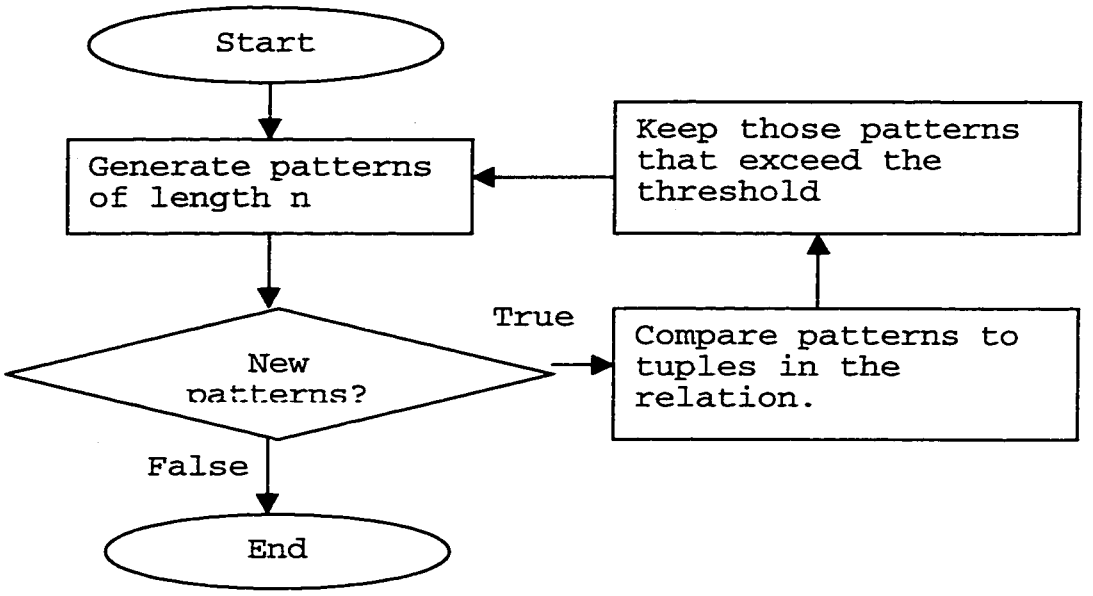


Figure 2. General Flowchart on Finding Association Rules

The combinations, or candidates, consist of one or more attribute values, (X_1, X_2, X_3, ... , X_n). As granules, the combination is an association rule if the bit count on the intersection of all the granules, $X_1 \cap X_2 \cap X_3 \cap \ldots \cap X_n$, is equal or greater than the minimal count, c (Lin 2000). The bit count is the number of 1's in the bit stream from the result of the intersection of the two or more granules. The bit representation of granules offers efficient storage while the intersection of granules offers fast computation in finding association rules.

Three columns are added to the vehicle relation: Number of Past Violations, Last Violation, and Last Accident. Each column has its own quotient set of attribute values, and each attribute value can be represented as granules. Table

6 shows the vehicle relation. Figure 3 shows the quotient sets for the three new columns in the vehicle relation.

Vehicle ID	*Vehicle Type*	*# Of Past Violation*	*Last Violation*	*Last Accident*
ID1	Sedan	0	Speeding	Yes
ID2	SUV	1	Red Light	Yes
ID3	Sedan	2	Red Light	No
ID4	Mini Van	2	Red Light	Yes
ID5	Mini Van	1	Speeding	Yes
ID6	Mini Van	5	Speeding	Yes
ID7	SUV	3	Tailgating	No
ID8	Sedan	2	Right-of-way	No
ID9	Sedan	1	Right-of-way	No
ID10	SUV	5	Tailgating	No
ID11	SUV	4	Speeding	Yes
ID12	Sedan	0	Red Light	No

Table 6: Vehicle relation #2

<u># of Past Violation Quotient Set</u>
0 = {ID1, ID12}
1 = {ID2, ID5, ID9}
2 = {ID3, ID4, ID8}
3 = {ID7}
4 = {ID11}
5 = {ID6, ID10}

<u>Last Violation Type Quotient Set</u>
Speeding = {ID1, ID5, ID6, ID11}
Red Light = {ID2, ID3, ID4, ID12}
Tailgating = {ID7, ID10}
Right-of-way = {ID8, ID9}

<u>Last Accident Quotient Set</u>
No = {ID3, ID7, ID8, ID9, ID10, ID12}
Yes = {ID1, ID2, ID4, ID5, ID6, ID11}

Figure 3.

From this table, one can conjecture that moving violations with mini vans and last accidents appear 25% of the data. The conjecture can be verified with the intersection of two granules, i.e. the bit representation of

the value, mini vans, from vehicle type and the value, yes, from last accidents. After the intersection, the result is counted. Similarly, other conjectures can be done with the same operations on multiple granules.

Below is the flowchart of discovering association rules using granules. The flowchart consists of the basic loop in figure 2.

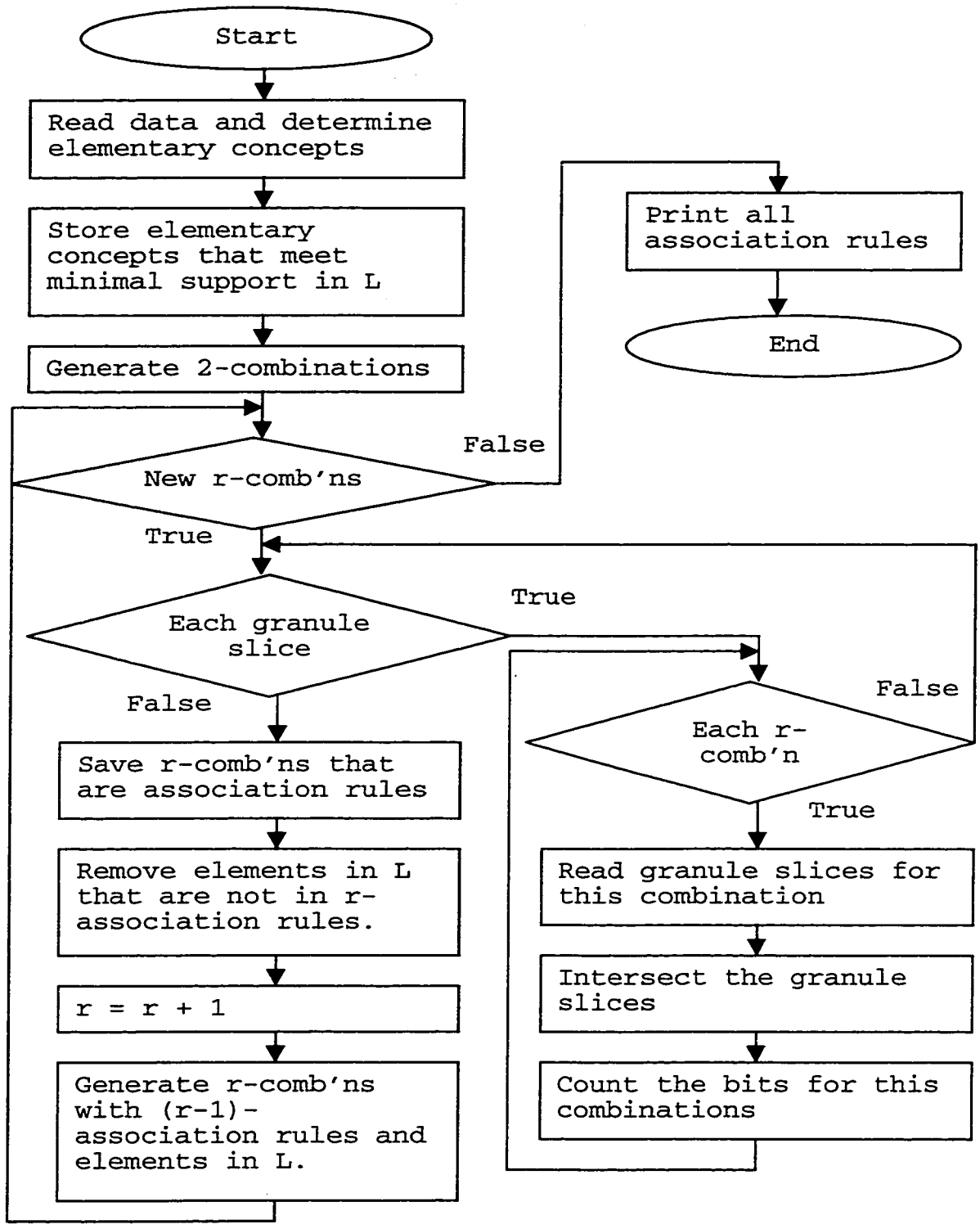


Figure 4. Flowchart to find association rules using granules

During each cycle, combinations of length r, or in short, r-combinations, are generated and are checked against the relation to determine which are r-association rules. When intersecting the granules, the size of the granules may be large. Thus, slices of the granules are read and processed until all slices of the granules are done. In the end, if the intersection of granules in the r-combination results in a count that meets or exceeds the minimal count, c, the combination is an association rule, and it is saved. At the end of the cycle, if any r-combinations are declared association rules, new combinations of length r+1 are generated for the next cycle. The cycle stops when no r-combinations are found to be association rules or if no new (r+1)-combinations can be generated from the discovered r-association rules. The next section explains how combinations are generated and how bit counting is done.

3.1. The Generation of Combinations

Creating combinations, or patterns, is a computational process of forming sets of attribute values. The number of combinations for one selected attribute is the size of the quotient set of that selected attribute. Each combination contains one attribute value from that selected attribute. The number of combinations for two selected attributes is the Cartesian product, $q_1 \times q_2$, where q_1 and q_2 are the sizes of the quotient set of the first and second selected

attributes respectively. Each combination contains two attribute values: one attribute value from the first and one attribute value from the second. For the general case, the number of r-combinations is $q_1 \times q_2 \times q_3 \times \ldots \times q_r$, where q_r is the size of the quotient set of each of the selected attributes. Each r-combination contains one attribute value from each selected attribute.

The more general case is to create r-combinations among the n attributes in the relation. Since no columns are selected beforehand, r unique attributes are chosen from the n attributes in the relation, and from those chosen r attributes, all combinations of attribute values from the domains of those attributes are formed. There are C(n,r) possible ways to choose r attributes from n attributes, and each possible way has its own set of r-combinations among the r attributes. In short, the number of r-combinations is the total sum of the each subtotal combination from each possible way to select r attributes.

Figure 3 shows the mathematical equations for the number of combinations of length 1, 2, and 3 with r attributes in the relation. The symbol, q, represents a specific attribute's quotient set size, and the summation symbol, Σ, appears before each term, q. The subscripts on each term, q, are the summation indexes to specify the particular attributes in the relation, so value of the

indexes ranges from 1 to n. The first summation index, g, has the range from 1 to n-(r-1). The second summation index, h, has the range from 2 to n-(r-2), and so on. Each subsequent summation index has the beginning and end indexes one more than the previous summation indexes.

$$\text{Combinations of Length 1} = \sum_{g=1}^{n} q_g$$

$$\text{Combinations of Length 2} = \sum_{g=1}^{n-1} \left(\sum_{h=g+1}^{n} (q_g q_h) \right)$$

$$= \sum_{g=1}^{n-1} \left(q_g \left(\sum_{h=g+1}^{n} q_h \right) \right)$$

$$\text{Combinations of Length 3} = \sum_{g=1}^{n-2} \left(\sum_{h=g+1}^{n-1} \left(\sum_{i=h+1}^{n} (q_g q_h q_i) \right) \right)$$

$$= \sum_{g=1}^{n-2} \left(q_g \left(\sum_{h=g+1}^{n-1} q_h \left(\sum_{i=h+1}^{n} q_i \right) \right) \right)$$

The general equation for the total number of r-combinations on n attribute in the relation is the following:

Number of Combinations of Length r =

$$\sum_{g=1}^{n-(r-1)} \left(q_g \left(\sum_{h=g+1}^{n-(r-2)} q_h \left(\sum_{i=h+1}^{n-(r-3)} q_i \left(\ldots \sum_{z=y+1}^{n} q_z \right) \right) \right) \right)$$

So far, the equation deals with various sizes of quotient sets on the columns in the relation. For the special case, where all attributes in the relation have the the same quotient sets size, K, the equation simplifies to the following: $\mathbf{C(n,r)K^r}$. Generally, the quotient set sizes for each attribute in the relation are not the same. Nonetheless, taking the average quotient set size of all the attributes may be useful to estimate the number of combinations of length r.

$$\mathbf{C(n,r)\left(\left(\sum_{g=1}^{n} q_g\right)/n\right)^r}$$

For a given relation with n columns, the maximum length of any combinations is the length n. For the relation, vehicle, the maximum length of any combinations is 4. Below show the quotient set sizes for the relation, vehicle, and the numbers for combinations of length r.

Quotient Set	Size
Vehicle Type	3
# of Past Violation	6
Last Violation	4
Accident	2

Table 7

Length	Estimate Combinations	Calculations	Actual Combinations
1	15	(3)+(6)+(4)+(2)	15
2	84.375	(3)(6)+(3)(4)+ (3)(2)+(6)(4)+ (6)(2)+(4)(2)	80
3	210.9375	(3)(6)(4)+(3)(6)(2)+ (3)(4)(2)+(6)(4)(2)	180
4	197.75390625	(3)(6)(4)(2)	144

Table 8

For additional clarity, all 80 combinations of length 2 are shown below. The elementary concepts are represented by two values: the attribute index and the value index. For example, (1 2) points to the second attribute value on the first attribute in the relation.

(1 1)(2 1), (1 1)(2 2), (1 1)(2 3), (1 1)(2 4), (1 1)(2 5),
(1 1)(2 6), (1 2)(2 1), (1 2)(2 2), (1 2)(2 3), (1 2)(2 4),
(1 2)(2 5), (1 2)(2 6), (1 3)(2 1), (1 3)(2 2), (1 3)(2 3),
(1 3)(2 4), (1 3)(2 5), (1 3)(2 6), (1 1)(3 1), (1 1)(3 2),
(1 1)(3 3), (1 1)(3 4), (1 2)(3 1), (1 2)(3 2), (1 2)(3 3),
(1 2)(3 4), (1 3)(3 1), (1 3)(3 2), (1 3)(3 3), (1 3)(3 4),
(1 1)(4 1), (1 1)(4 2), (1 2)(4 1), (1 2)(4 2), (1 3)(4 1),
(1 3)(4 2), (2 1)(3 1), (2 1)(3 2), (2 1)(3 3), (2 1)(3 4),
(2 2)(3 1), (2 2)(3 2), (2 2)(3 3), (2 2)(3 4), (2 3)(3 1),
(2 3)(3 2), (2 3)(3 3), (2 3)(3 4), (2 4)(3 1), (2 4)(3 2),
(2 4)(3 3), (2 4)(3 4), (2 5)(3 1), (2 5)(3 2), (2 5)(3 3),
(2 5)(3 4), (2 6)(3 1), (2 6)(3 2), (2 6)(3 3), (2 6)(3 4),
(2 1)(4 1), (2 1)(4 2), (2 2)(4 1), (2 2)(4 2), (2 3)(4 1),
(2 3)(4 2), (2 4)(4 1), (2 4)(4 2), (2 5)(4 1), (2 5)(4 2),
(2 6)(4 1), (2 6)(4 2), (3 1)(4 1), (3 1)(4 2), (3 2)(4 1),
(3 2)(4 2), (3 3)(4 1), (3 3)(4 2), (3 4)(4 1), (3 4)(4 2)

For the relation, vehicle, the total number of combinations of all lengths is 15 + 80 + 180 + 144 = 419.

For a relation with many attributes or attribute values, the number of combinations can be very large. Each combination requires a count of the number of tuples in the

relation that the combination contains. The combinations with the count greater than the minimal support are association rules. The next section demonstrates methods to reduce the number of combinations. In doing so, the number of comparisons between the combinations and the tuples in the relation are saved.

3.2 The Generation of Potential Combinations

The number of combinations is an important factor to consider in the process. Each combination has the granules in the combination intersected and the result counted. If a combination does not have the potential of becoming association rule, the combination should not be undergoing this process. So only potential combinations are generated in the process. One important factor is that all subsets of an association rule are also association rules. In other words, the subsets of an association rule, A, have at least the same support count as association rule A.

All subsets of an association rule are association rules.

Proof: Let the association rule be called A. Let subset of association rule, A, be called B, and let the remaining subset, A-B, be called C. Let B be the subset that is not an association rule. Then, the intersection of the set of supported tuples for B and the set of supported tuples for C

should be enough so that A can be an association rule. This is not possible since an intersection among sets could only result in the size of a smaller set. Since B has the smaller set of support tuples and is not an association rule, the intersection of the B and C could not result in a set that is an association rule. Thus, it must be true that all subsets of an association rule are also association rules.

Therefore, all discovered r-association rules are derived from previously discovered s-association rules, where the value of s is smaller than r. This is the basis for minimizing the number of combinations.

Each new r-combination must have all its subsets, except for the combination itself, be association rules; otherwise, the r-combination is removed. The following two rules are used to generate r-combinations:

1) Two association rules are combined to generate a new r-combination. An r-combination is made from the two association rules: one is an (r-1)-association rule and the other is 1-association rule which single element in the 1-association rule is not of any element in (r-1)-association rule.
2) All new r-combinations have all (r-1)-subsets an association rule. The previous rule does not always create an r-combination that has all subsets an

association rule. Therefore, every (r-1)-subsets of the new combinations is checked. For example, the 4-combination, {A, B, C, D}, is a potential combination if all 3-subset combinations, {A, B, C}, {A, B, D}, {A, C, D}, and {B, C, D}, are association rules. If any of these subsets were not association rules, then the combination, {A, B, C, D}, will never be an association rule. Note: if all (r-1)-subsets are an association rules, then all subsets of any length less than r-1 are association rules.

The algorithm starts with a list, L, which contains elementary concepts that are also 1-association rules. When making r-combinations, all the elementary concepts in the list, L, are verified if they exist as elements in any (r-1)-association rules. If an elementary concept does not exist in any (r-1)-association rules, it is removed from the list. Next, new r-combinations are created from the (r-1)-association rule and the list, L, by joining an (r-1)-association rule with element in L that has an attribute index greater than all attribute indexes of elements in that (r-1)-association rule. Only the new r-combinations that have every (r-1)-subset an association rule are kept.

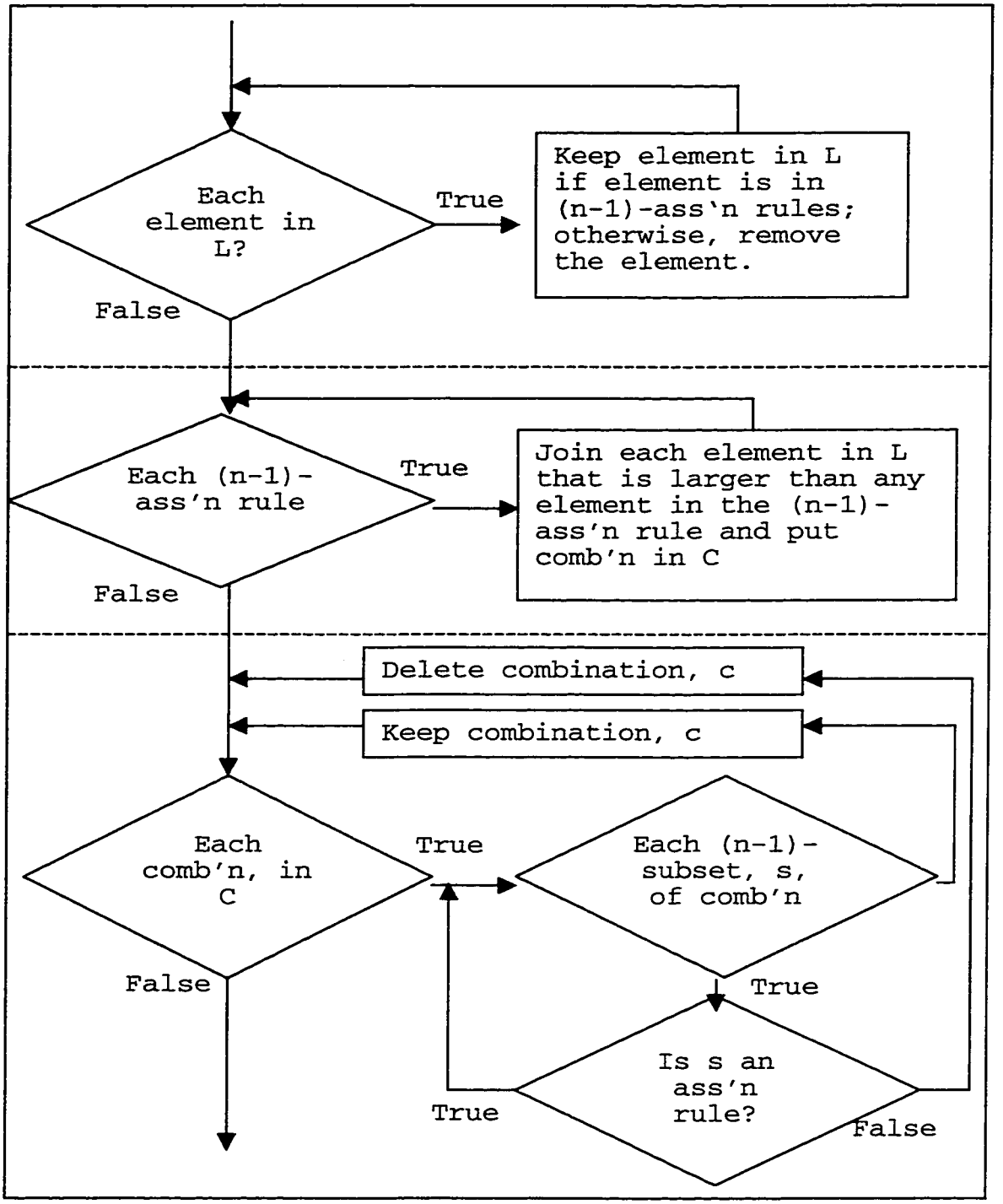


Figure 5. Flowchart to create new combinations

Below is an example of creating 4-combinations from 3-association rules.

Let the following be the current 3-association rules.

{(6 0) (11 0) (12 0)}, {(6 0) (11 0) (14 2)},
{(6 0) (11 0) (15 2)}, {(6 0) (11 1) (12 3)},
{(6 0) (11 1) (14 2)}, {(6 0) (11 1) (15 2)},
{(6 0) (14 2) (15 2)}, {(6 1) (11 0) (12 0)},
{(6 1) (11 1) (12 3)}, {(11 0) (14 2) (15 2)},
{(11 1) (14 2) (15 2)}

And let the following be the elements in the list, L,

{(4 0)}, {(4 10)}, {(6 0)}, {(6 1)}, {(11 0)},
{(11 1)}, {(12 0)}, {(12 3)}, {(14 2)}, {(15 2)}

First, the elementary concepts, {(4 0)} and {(4 10)}, are removed from the list, L. Any 4-combinations that include these elementary concepts would not be an association rule. Below are the remaining elementary concepts in the list, L.

{(6 0)}, {(6 1)}, {(11 0)}, {(11 1)}, {(12 0)},
{(12 3)}, {(14 2)}, {(15 2)}

Next, each 3-association rule is combined with elements in L to create 4-combinations, if possible. This 3-association rule, {(6 0) (11 0) (12 0)}, is combined with these two elements in L: {(14 2)} and {(15 2)}. This 3-association rule, {(6 0) (11 0) (14 2)}, is combined with one element in L, {(15 2)}, and so on with each 3-association rule. All the 4-combinations are shown below.

{(6 0) (11 0) (12 0) (14 2)} {(6 0) (11 0) (12 0) (15 2)}
{(6 0) (11 0) (14 2) (15 2)} {(6 0) (11 1) (12 3) (14 2)}
{(6 0) (11 1) (12 3) (15 2)} {(6 0) (11 1) (14 2) (15 2)}
{(6 1) (11 0) (12 0) (14 2)} {(6 1) (11 0) (12 0) (15 2)}

{(6 1) (11 1) (12 3) (14 2)} {(6 1) (11 1) (12 3) (15 2)}

As the 4-combinations are made, they are checked if all (r-1)-subsets are association rules. The combination, {(6 0) (11 0) (12 0) (14 2)}, is removed because this subset, {(11 0) (12 0) (14 2)}, is not an association rule. The combination, {(6 0) (11 0) (12 0) (15 2)}, is removed because this subset, {(11 0) (12 0) (15 2)}, is not a 3-association rule. The checks are done for each 4-combination. From eleven 4-combinations are produced, only two 4-combinations are kept. Below are the 4-combinations. All of these combinations have the potential of becoming association rules.

{(6 0)(11 0)(14 2)(15 2)}
{(6 0)(11 1)(14 2)(15 2)}

The next section determines which 4-combinations are association rules.

3.3. The Cost from Using Granules

For each r-combination, the granules in the combinations are intersected and the result is counted. If the granules are represented in list form, the intersection requires the comparisons of the tuple name among the granules. If the granules are represented in bit form, the intersection of granules in bit form requires the bit-wise AND operation among the granules. The bit-wise operation is

done using the machine's word size operations. The number of the AND instructions on one r-combination is shown below:

$$\text{AND operations} = \left\lceil \frac{\text{Number of tuples}}{\text{Bits / word}} \right\rceil * (r-1)$$

For example, the 4-combination consists of 4 granules. If the relation has 20480 tuples, then there are 640 words (20480/32 = 640) in a granule. Between four granules, there are 3 AND operations required. Thus, a total of 1920 AND operations (20480/32 * 3 = 1920) are needed to perform the intersection of the four granules.

After the AND operations, the bits in the result are counted. Below is the method of counting the bits in the result. The bits are counted by repeatedly dividing the word or subtotal by 2^x and adding the first half with second half on each division (Louie 491). The letter, x, begins with 1 and increments on each cycle until 2^x equals the length of the subtotal. The bit count of the result is done word size at a time. First, each pair of bits in a word is counted by adding the even bits with the odd bits. This step has 16 subtotals. Then, every adjacent pair of the subtotals is added together, resulting in half the number of subtotals from the previous step. This step is repeated until the number of subtotals results in one total. For example, the word, 3C 5E 0F 79, has the following binary representation:

0011 1100 0101 1110 0000 1111 0111 1001.

Below show the bits being counted. Each subtotal is delimited with the symbol, |. The bits within the delimiter are shown.

```
0x|1x|1x|0x|0x|0x|1x|1x|0x|0x|1x|1x|0x|1x|1x|0x <= even bits
x0|x1|x1|x0|x1|x1|x1|x0|x0|x0|x1|x1|x1|x1|x0|x1 <= odd bits
00|10|10|00|01|01|10|01|00|00|10|10|01|10|01|01 <= subtotals

00xx|10xx|01xx|10xx|00xx|10xx|01xx|01xx <= even subtotals
xx10|xx00|xx01|xx01|xx00|xx10|xx10|xx01 <= odd subtotals
0010|0010|0010|0011|0000|0100|0011|0010 <= eight subtotals

0010xxxx|0010xxxx|0000xxxx|0011xxxx <= even subtotals
xxxx0010|xxxx0011|xxxx0100|xxxx0010 <= odd subtotals
00000100|00000101|00000100|00000101 <= four subtotals

00000100xxxxxxxx|00000100xxxxxxxx <= even subtotals
xxxxxxxx00000101|xxxxxxxx00000101 <= odd subtotals
0000000000001001|0000000000001001 <= two subtotals

00000000000001001xxxxxxxxxxxxxxxx <= even subtotal
xxxxxxxxxxxxxxxx0000000000001001 <= odd subtotal
00000000000000000000000000010010 <= one totals
```

The result is a binary value, 18, which is the number of bits SET for the word, 3C 5E 0F 79.

Below shows the sample C code to count bits SET.

```
  unsigned long bits;
  unsigned long Count32;
  unsigned SubTotalCount;
        . . .
  Count32 = (bits&0x55555555)+((bits>>1)&0x55555555);
  Count32 = (Count32&0x33333333)+((Count32>>2)&0x33333333);
  Count32 = (Count32&0x0F0F0F0F)+((Count32>>4)&0x0F0F0F0F);
  Count32 = (Count32&0x00FF00FF)+((Count32>>8)&0x00FF00FF);
  SubTotalCount += (Count32&0x0000FFFF)+(Count32>>16);
```

The first four assignment statements take 28 instructions, 4 * (3 *move* + 2 *and* + 1 *add* + 1 *shr*), and the last extended

assignment statement takes 7 instructions, (3 *move* + 1 *add* + 2 *add*, + 1 *shr*). So the total count for one 32-bit word is 35 instructions. The advantages with this method are the cost is linear and there are no control statements. So, the total cost to check an r-combination is the following:

$$\text{Total instruction} = \left\lceil \frac{\text{Number of tuples}}{\text{Bits / Word}} \right\rceil * (r-1 + 35)$$

For example, with 5 granules in the combination and 2^{20} tuples in the relation, the instruction cost is the following:

$$\frac{2^{20}}{2^5} * ((5-1) + 35) = 2^{15} * (39) = 1{,}277{,}952 \text{ inst.}$$

3.4. The Storage and Management of Granules

The granules are stored on disk in data pages. The data pages offer flexibility in accessing and processing the granules. Portions of the granules, called slices, are stored in the data pages, and the data pages are connected by links. Below shows an example of several data pages holding three granules. Each granule has its initial page on a different page. The initial pages for each granule are not necessarily consecutive.

Granule	Initial Page
A	1
B	2
C	5

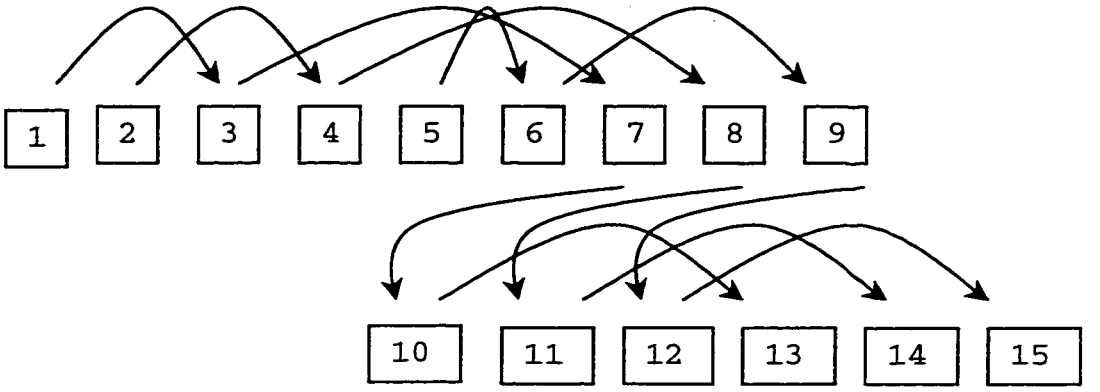


Figure 6.

Each granule requires the same number of data pages to store the binary representation of the list, and for this example, each granule has five data pages. The arrows in the figure 6 indicate the links to the next data page for the granules.

For the combination, {A, B, C}, the granules for A, B, and C are intersected, and the result is counted. Since the granules are stored in data pages, the data pages 1, 2, and 5 are intersected first, and the bits in the result are counted. Next, pages, 3, 4, and 6 are intersected, and the bits in this intersection are counted. This subtotal is added to the previous subtotal. This continues on for the third, forth, and fifth pages of the granules A, B, and C. After the last page, the combination, {A, B, and C} is an association rule if the final total is greater than the minimal support.

Throughout the intersection and bit counting, many data pages are read among the r-combinations in each cycle. Even though the granules in each r-combination do not repeat

within the r-combination, the granules in one r-combination may appear among other r-combinations. This means that data pages read for one combination may be needed for other combinations. So to reduce the physical data page reads for granules, all the nth data pages on each granule from each r-combination are processed at the same time. In other words, all the nth data pages on each combination are intersected, the results of the intersections are counted, and the counts are added to each combination's subtotal before continuing to the next data pages on each combination. As a result, the data pages for each granule are read once from disk to memory for r-combinations in determining which are association rules.

The total storage cost for granules is based on the number of granules in the relation and the number of data pages per granule. The number of pages per granule is dependent on the number of tuples in the relation and the number of tuples that can be stored per page. The equation for the total storage cost is the following:

$$\text{Storage cost} = \text{Number of granules} * \left\lceil \frac{\text{Number of tuples}}{\text{Max tuples/page}} \right\rceil * \text{Data page size}$$

For example, the relation has 1,000 granules and 100,000 tuples. Each data page is 4096 bytes, and 4080 out of 4096 bytes are available to store data for granules. Then, the following is the total storage costs:

$$\text{Storage cost} = 1000 * \left\lceil \frac{100000}{4080 * 8} \right\rceil * 4096$$

$$= 16384000 \text{ bytes}$$

4. Using Apriori to Find Association Rules

The algorithm, Apriori, by Rakesh Agrawal follows the same loop as in figure 2. The algorithm uses the concept of finding new knowledge from previous knowledge. The algorithm generates r-candidates from (r-1)-association rules, and compares by value the r-candidates to tuples in the relation. The latter part is what the data model of the granules improves on. Below explains the algorithm, Apriori, in further detail.

4.1. The Generating of Candidates

The algorithm, Apriori, generates the r-candidates by combining two (r-1)-association rules that have the first r-2 attribute values in the two (r-1)-association rules the same and the last pair does not. The new r-candidate has all attribute values from the first association rule plus the last attribute value from second association rule. For g (r-1)-association rules, (g-g)g/2 attempts are made to generate new r-candidates.

Once an r-candidate is made, the r-candidate must meet the second criterion from Section 3.2:

> *All new combinations generated must have all (r-1)-subsets an association rule.*

A new candidate is kept if every (r-1)-subset is an association rule. If any subsets are not association rules, then this new candidate is removed. This step in Apriori is referred to as the prune step. These two steps are the significant contribution from Apriori, for it reduced the number of r-combination to check against the relation. The algorithm, granule, follows the same concept; therefore, the final list of r-candidates between the two algorithms are the same.

4.2 The Determination of Association Rules

Each r-candidate generated poses a simple aggregation query against the relation. For instance, if an r-candidate contains the following attribute values {Vehicle Type = 'mini van', Violation = 'Red Light', Accident = 'Yes'}, then that particular candidate forms the following SQL query:

```
SELECT
   CASE
      WHEN COUNT (*) >= Minimal Support THEN
           'This candidate is an association rule.'
      ELSE 'This candidate is not an association rule'
   END
   FROM Table name
   WHERE Vehicle Type = 'mini van' AND
                Violation = 'Red Light' AND
                Accident = 'Yes'
```

The SQL query evaluates each predicate on each tuple in the relation and counts the tuples that satisfy all predicates. To evaluate an r-candidate, a relation scan is required.

The algorithm, Apriori, evaluates all r-candidates at once, so only one relation scan is done for all r-candidates. On each tuple in the relation, comparisons are made for each attribute value in the r-candidates to the tuple. For those r-candidates that have every attribute value in the tuple, the counters in those r-candidates are incremented. In all, one relation scan is required for one set of r-candidates. Even with an index, any one r-candidates would require a relation scan.

4.3 The Subset Function

For p candidates and q tuples in the relation, p x q comparisons are needed to determine the count for the p candidates. A hash-tree table is used to reduce the comparisons. The node of the hash-tree table is either a leaf node containing some candidates or an interior node containing a hash table (Agrawal 502). The hash table consists of references to other interior nodes or leaf node (Agrawal 502). Below shows a hash-tree table. The attribute values are displayed in the format, (a b), representing the attribute index and the attribute value respectively.

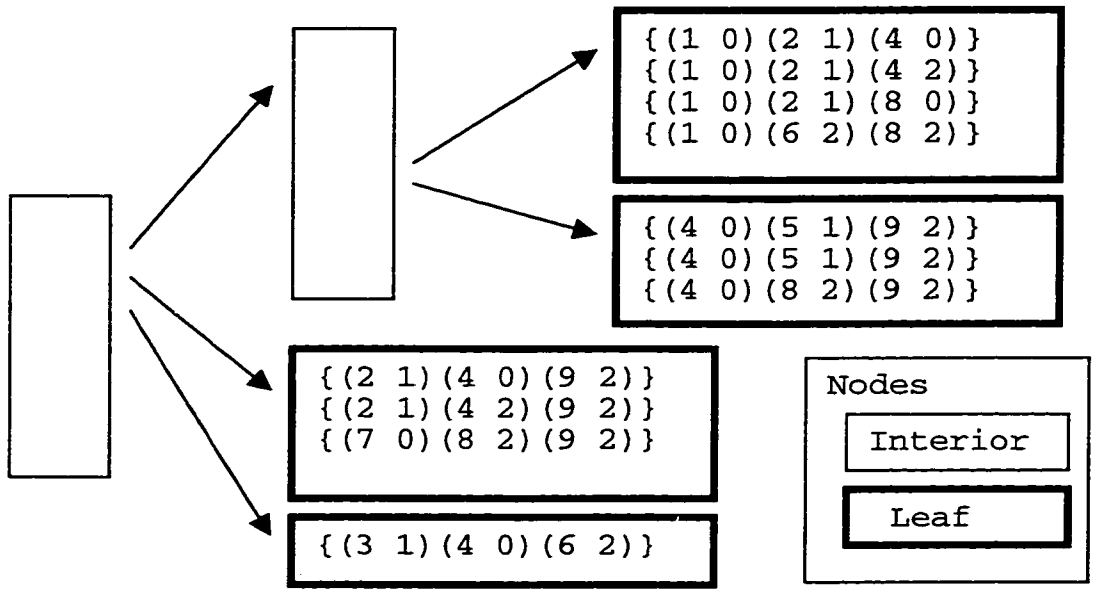


Figure 7. Hash Table

The hash-tree table reduces the candidates to compare on each tuple by dividing the p candidates among the leaf nodes. All the p candidates have the same length, r. Each tuple in the relation has C(n,r) subsets, and each subset is hashed on the hash-tree table. For subsets that hash onto a non-visited leaf node, each candidate in those leaf nodes is compared to the tuple. The subsets that hash to a visited leaf nodes or interior node are skipped. This is the subset function that is applied each tuple in the relation. The cost of the subset function is determined by two factors: the number of subsets per tuple and the number of candidates for all visited-once nodes per tuple. Below shows the equation:

$$m(C(n,r)) + \sum_{i=1}^{m}(v_i * t)$$

The first term is the costs of calling the hash function for m tuples in the relation. The second term is the sum cost of each tuple's comparisons. The letter, v_i, is the number of visited-once leaf nodes per tuple, and the letter, t, is the average number of candidates per leaf nodes. The letter, m, is the number of tuples in the relation, the letter, n, is the number of attributes in the relation, and the letter, r, is the current length of the candidates in the hash-tree table. Below is an example of how the subset function is applied to this tuple in the relation:

{(1 5), (2 5), (3 6), (4 8), (5 9), (6 7)}

A hash-tree table with 3-candidates has this tuple hashed with C(6,3) 3-subsets. Below are the all the subsets hashed on the hash-tree table.

1: (1 5)(2 5)(3 6)
2: (1 5)(2 5)(4 8)
3: (1 5)(2 5)(5 9)
4: (1 5)(2 5)(6 7)
5: (1 5)(3 6)(4 8)
6: (1 5)(3 6)(5 9)
7: (1 5)(3 6)(6 7)
8: (1 5)(4 8)(5 9)
9: (1 5)(4 8)(6 7)
10: (1 5)(5 9)(6 7)
11: (2 5)(3 6)(4 8)
12: (2 5)(3 6)(5 9)
13: (2 5)(3 6)(6 7)
14: (2 5)(4 8)(5 9)
15: (2 5)(4 8)(6 7)
16: (2 5)(5 9)(6 7)
17: (3 6)(4 8)(5 9)
18: (3 6)(4 8)(6 7)
19: (3 6)(5 9)(6 7)
20: (4 8)(5 9)(6 7)

For the subsets that hash to a non-visited leaf node, the candidates in the leaf node are compared with the tuple. The number of comparisons between candidates and tuple is

the number of visited leaf nodes times the average number of candidates per leaf node. On the 20 subsets above, 15 of subsets may hash onto non-visited leaf node, 3 of subsets may hash onto visited leaf node, and 2 of subsets may hash onto leaf node. If there are 1000 3-candidates and the average candidates per leaf node is 30, then the number of comparisons for that tuple is 15 * 30 = 450. This is far less than 1000. Less comparisons for each tuple is likely when the r-candidates are distributed evenly among the leaf nodes in the hash-tree table and the number of subsets per tuple is small.

Apriori is improved further by short-circuiting the comparisons on each candidate to each tuple and by comparing the r-candidates to a (r-1)-candidate bar table consisting of (r-1)-candidates ids. The first improvement has the comparisons stop once the first element in the r-candidate is not contained in the tuple. The second improvement uses the (r-1)-candidate bar table to compare only the (r-1)-candidate ids of the tuples. Basically, when an r-candidate is compared to a tuple in the (r-1)-candidate bar table, the two (r-1)-candidate ids, that generated the r-candidate, is compared to the (r-1)-candidate ids in the tuple. If both (r-1)-candidate ids exist for that tuple, the r-candidate id for this r-candidate is inserted into the r-candidate bar table for that tuple. This is explained in detail in "Fast

Algorithms for Mining Association Rules." The use of the candidate bar table depends on the number of (r-1)-candidate ids per tuple. If the number of (r-1)-candidate ids per tuple is high, the cost of creating the (r-1)-candidate bar table outweighs its benefit. Therefore, Rakesh has the algorithm, AprioriHybrid. The algorithm has the normal comparisons on the relation in the first three cycles, and then it switches to the comparisons on the candidate bar table. Below is the flowchart for the algorithm, AprioriHybrid.

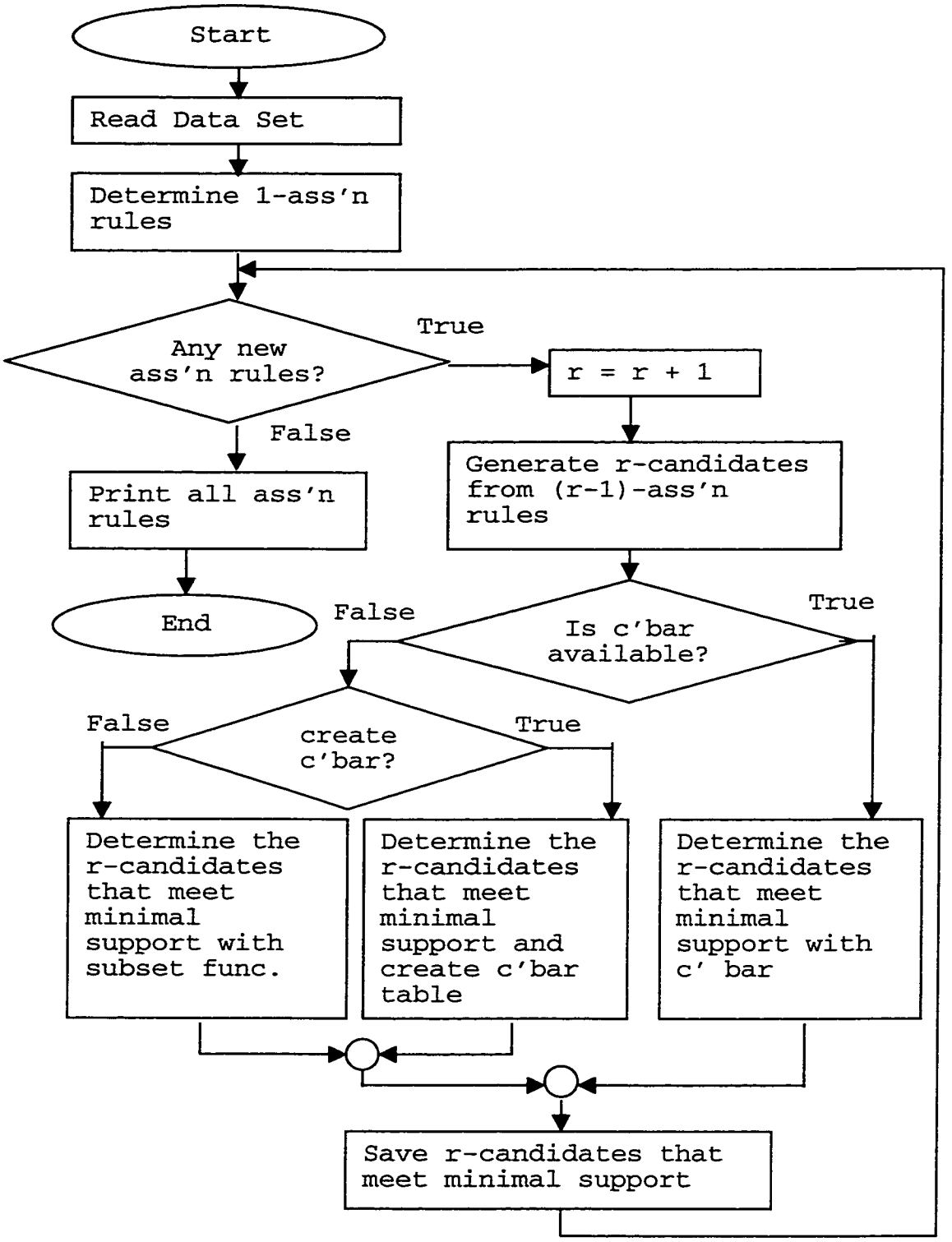


Figure 8. Finding association rules

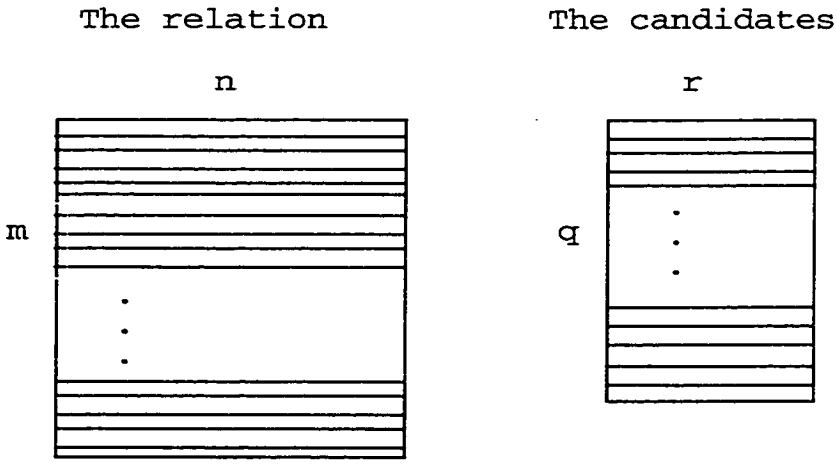
The next section compares the two algorithms, Apriori and Granule.

5.0 Comparison between Apriori and Granule

The algorithms, Apriori and granule, are compared based on their key operations. For the algorithm, granule, the number of AND operations between the granules determines the cost. For the algorithm, Apriori, the number of comparisons between the attribute values in the candidates and the tuples determines the cost. A ratio is used to compare the two algorithms, Apriori and Granule.

$$\text{Let } x = \frac{\text{Number of comparisons operations in Apriori}}{\text{Number of AND operations in Granules}}$$

If the ratio is less then 1, then algorithm, Apriori, is favored to perform fast. If the ratio is greater than 1, then the algorithm, granule, is favored to perform fast.



Let m = the number of tuples in a relation
Let n = the number of attributes in a relation
Let r = the number of attribute values per candidates.
Let q = the number of the q-candidates
All values for m, n, q, and r are positive integers.

Figure 9. The relation and candidates

The worse case for the algorithm, Apriori, is m * (r * q) comparison operations. Each candidate compares with each tuple to determine if it is contained in the tuple. The worse case for algorithm, granules is (m/32) * (r-1) * q AND operations. Each candidate needs m/32 * (r-1) AND operations. So, the ratio is the following:

$$x = \frac{m * (r * q)}{(m/32) * (r-1) * q} = \frac{32 * r}{r - 1}$$

The granule method can execute 32 times faster than Apriori, and there is no positive integer for r that makes the ratio less than 1.

The subset function in Apriori lowers the costs if the subset function reduces the number of r-candidates to compare to each tuples in the relation. However, the cost varies per tuple in the relation and is affected by the number of attribute in the relation. Yet, the subset function favors Apriori if the overall cost of the subset function reduces the numerator by 1/32 or 0.03125.

5.1 Sample Runs using Apriori and Granule

Four different data are generated to compare the run time on these algorithms to find association rules. The

data varies the number of tuples, the attributes per relation, and the minimal support. Each attribute ranges from 2 to 100 attributes.

Data	Rows	Columns	Table Size	Granule Size	Memory	Minimal Support
1	100000	16	6.4M	7.7M	10M	2000
2	400000	16	25.6M	36.7M	10M	8000
3	100000	48	19.5M	27.8M	10M	2000
4	400000	48	78.0M	88.0M	10M	8000

Table 9.

The first data set has a smaller data size than the memory. It characterizes a memory-based operation. The other three data set have a larger data size than memory. The algorithms to compare are the following: Granule, Apriori, Apriori with subset, and AprioriHybrid. The algorithm, granule, generates r-combinations by combining (r-1)-association rules with 1-association rules. All versions of Apriori, generates r-candidates by combining two (r-1)-associations rules.

Below show the runs for data 1 and 2. The first section displays the quotient set for each column in the relation. The second section shows the number of r-candidates and r-association rules. Since two methods are used to create r-candidates, both are shown. The third section shows the individual execution times for finding r-association rules. The execution times include the times to create r-candidates and the times to determine the r-

association rules from the r-candidates. The last section displays the number of real and logical reads for each algorithm.

```
Data 1: columns IDs, column names, and quotient sets
 0:  C1  7   4:  C5 28   8:  C9 51  12: C13 23
 1:  C2 53   5:  C6 10   9: C10 17  13: C14 25
 2:  C3  2   6:  C7 31  10: C11 62  14: C15 53
 3:  C4 15   7:  C8 24  11: C12  8  15: C16 62
```

Granules numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	471	0	471	291
Length 2 :	39168	0	39168	214
Length 3 :	8676	8628	48	14
Length 4 :	55	55	0	0
Total :	48370	8683	39687	519

Apriori numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	471	0	471	291
Length 2 :	39168	0	39168	214
Length 3 :	6070	6022	48	14
Length 4 :	0	0	0	0
Total :	45709	6022	39687	519

Length	Candidates	Granule	Apriori	Apriori Subset	Apriori Hybrid
1	471	0.00	0.00	0.00	0.00
		1.84	0.94	0.94	0.94
2	39168	0.09	0.34	0.33	0.33
		14.36	770.16	104.91	104.97
3	48	0.17	0.14	0.13	0.14
		0.02	0.33	6.49	8.22
4	0	0.02	0.00	0.00	0.00
		0.00	0.00	0.00	0.00
		16.50	771.91	112.78	114.59
Real Reads		3048	1563	1563	1563
Logical Reads		312756	3126	3126	3126

```
Data 2: column IDs, column names, and quotient sets
 0:  C1 54   4:  C5 34   8:  C9 52  12: C13 37
 1:  C2 37   5:  C6 54   9: C10 18  13: C14 41
 2:  C3 53   6:  C7 51  10: C11 53  14: C15 41
 3:  C4 57   7:  C8 56  11: C12 16  15: C16 35
```

Granules numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	689	0	689	307
Length 2 :	43556	0	43556	4
Length 3 :	2	1	1	1
Length 4 :	0	0	0	0
Total :	44247	1	44246	312

Apriori numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	689	0	689	307
Length 2 :	43556	0	43556	4
Length 3 :	1	0	1	1
Length 4 :	0	0	0	0
Total :	44246	0	44246	312

Length	Candidates	Granule	Apriori	Apriori Subset	Apriori Hybrid
1	689	0.00	0.00	0.00	0.00
		8.75	4.06	3.84	3.84
2	43556	0.13	0.39	0.39	0.41
		51.95	3400.28	369.19	369.20
3	1	0.00	0.00	0.00	0.00
		0.00	0.44	24.86	26.83
4	0	0.00	0.00	0.02	0.00
		0.00	0.00	0.00	0.00
		60.83	3405.17	398.30	400.28
Real Reads		12963	18750	18750	18750
Logical Reads		1128489	0	0	0

Here are some observations and explanations on the results.

1) All versions of Apriori use the same routine to find 1-association rules; therefore, all execution times are the same for all versions of Apriori on finding 1-association rules.
2) The routine, granule, creates a temporary table before determining 1-association rules, so its run time to find 1-association rules is longer than Apriori.

3) AprioriHybrid uses the routine, AprioriSubset, to determine 2-association rules and 3-association rules. The run time to find 2 and 3-association rules are similar.
4) On data 1 and 2, AprioriSubset wins over Apriori when finding 2-association rules because the subset function in AprioriSubset reduces the number of 2-candidates to compare to each tuple in the relation.
5) On finding 3-association rules on AprioriSubset, the number of subsets per tuple is many times the number of 3-candidates. Each tuple in data 1 and 2 has C(16,3) and C(48,3) subsets respectively. Finding 3-association rules with Apriori is better than AprioriSubset.
6) AprioriHybrid needs the 3-candidate bar tables when finding 4-association rules. So the 3-candidate bar table is created while finding 3-association rules. As a result, the execution on finding 3-association rules in AprioriHybrid is longer than AprioriSubset.
7) On data 1, the table and granule sizes are smaller than the memory size. Finding association rules for data 1 is memory-based.
8) On data 1, all version of Apriori requires 1563 real reads to do one relation scan. All subsequent relation scans are logical reads.
9) On data 2, all version of Apriori requires 18750 real reads for the three cycles. Each table scan replaces

the memory twice since the stored data is 2½ times larger than the memory size.

10) On data 1, the granule performs 6.8 times faster than AprioriSubset.

11) On data 2, the granule performs 6.5 times faster than AprioriSubset.

Next show the runs for data 3 and 4.

Data 3: columns IDs, column names, and quotient sets

0:	C1	18	10: C11	47	20: C21	11	30: C31	8	40: C41 13
1:	C2	24	11: C12	27	21: C22	62	31: C32	58	41: C42 24
2:	C3	18	12: C13	56	22: C23	41	32: C33	35	42: C43 58
3:	C4	39	13: C14	11	23: C24	11	33: C34	43	43: C44 35
4:	C5	65	14: C15	50	24: C25	21	34: C35	6	44: C45 51
5:	C6	47	15: C16	45	25: C26	23	35: C36	42	45: C46 26
6:	C7	61	16: C17	54	26: C27	9	36: C37	42	46: C47 2
7:	C8	5	17: C18	59	27: C28	30	37: C38	15	47: C48 40
8:	C9	59	18: C19	37	28: C29	66	38: C39	59	
9:	C10	46	19: C20	34	29: C30	52	39: C40	10	

Granules numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	1695	0	1695	820
Length 2 :	327899	0	327899	785
Length 3 :	21909	20568	1341	51
Length 4 :	16	16	0	0
Total :	351519	20584	330935	1656

Apriori numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	1695	0	1695	820
Length 2 :	327899	0	327899	785
Length 3 :	3617	2276	1341	51
Length 4 :	0	0	0	0
Total :	333211	2276	330935	1656

Length	Candidates	Granule	Apriori	Apriori Subset	Apriori Hybrid
1	1695	0.00	0.00	0.00	0.00
		9.02	4.67	3.16	3.16
2	327899	4.67	17.69	17.63	17.75
		130.91	6452.06	1776.98	1776.00
3	1341	2.11	0.42	0.42	0.41

		0.55	13.75	396.06	408.61
4	0	0.01	0.00	0.00	0.01
		0.00	0.00	0.00	0.00
		147.27	6488.59	2194.25	2205.94
Real Reads		10153	13582	13582	13582
Logical Reads		2635911	704	704	704

Data 4: column IDs, column names, and quotient sets

0: C1 11	10: C11 13	20: C21 43	30: C31 11	40: C41 41
1: C2 48	11: C12 39	21: C22 58	31: C32 11	41: C42 54
2: C3 36	12: C13 9	22: C23 60	32: C33 61	42: C43 20
3: C4 36	13: C14 58	23: C24 61	33: C34 15	43: C44 56
4: C5 53	14: C15 3	24: C25 18	34: C35 11	44: C45 47
5: C6 17	15: C16 21	25: C26 54	35: C36 31	45: C46 19
6: C7 2	16: C17 32	26: C27 6	36: C37 50	46: C47 15
7: C8 57	17: C18 47	27: C28 31	37: C38 37	47: C48 18
8: C9 65	18: C19 28	28: C29 62	38: C39 13	
9: C10 46	19: C20 59	29: C30 43	39: C40 28	

Granules numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	1654	0	1654	957
Length 2 :	446298	0	446298	1200
Length 3 :	183700	181893	1807	176
Length 4 :	5130	5120	10	1
Length 5 :	0	0	0	0
Total :	636782	187013	449769	2334

Apriori numbers

	Candidates	Pruned Candidates	Final Candidates	Final Association
Length 1 :	1654	0	1654	957
Length 2 :	446298	0	446298	1200
Length 3 :	85093	83286	1807	176
Length 4 :	1508	1498	10	1
Length 5 :	0	0	0	0
Total :	534553	84784	449769	2334

Length	Candidates	Granule	Apriori	Apriori Subset	Apriori Hybrid
1	1654	0.00	0.00	0.00	0.00
		60.34	28.44	28.69	36.56
2	446298	9.33	30.03	30.03	29.95
		568.84	35280.64	9699.50	9652.92
3	1807	26.22	14.97	14.77	14.75
		10.80	80.63	1587.61	1708.94
4	10	0.16	0.05	0.08	0.06
		0.01	29.34	7375.17	5.64

5	0	0.00	0.00	0.09	0.00
		0.00	0.00	0.00	0.00
		675.70	35464.09	18735.94	11448.83
Real Reads		35662	76192	76192	57144
Logical Reads		11660581	0	0	0

Here are observations and explanations on these last two data sets.

12) The run times to generate new r-candidates or r-combinations are typically smaller than the run times to process the r-candidates to the relation.

13) For all version of Apriori, the logical reads in data 3 appear because the stored data is less than 2 times the size of memory. The table size is not large enough to flush the memory on one relation scan.

14) AprioriHybrid on data 4 took 5.64 seconds to find 4-association rules. This shows the benefits with the candidate bar table compared to Apriori and AprioriSubset.

15) The granule has a high number of logical reads because of the overlapped of attribute values among the r-combinations.

16) In data 4, AprioriHybrid has 57144 real reads that is 19048 less than the real reads for Apriori and AprioriSubset. The difference is the number of reads for one relation scan. AprioriHybrid did not perform a relation scan when finding 4-association rules.

Instead, it used the candidate bar table. The reads on the candidate bar table are not shown here.

17) On data 3, the granule performs 14.9 times faster than AprioriSubset.

18) On data 4, the granule performs 16.4 times faster than AprioriHybrid.

19) The granule size is calculated by the following: granule size = total number of granules per table * number of pages per granule * page size. The number of granules per table is the number of 1-candidates. The number of pages per granule is approximately the number of tuples in a relation divided by number of tuples per page. Finally, the page size is 4096.

20) The memory size in granule is greater than the page size * 1-candidates. When this is true, the number of real reads is low and the number of logical reads is high. When this is false, some data pages in memory are more likely to be replaced with another data pages. As a result, logical reads are replaced with real reads.

21) The granule's number of reads can be verified with the following equation:

$$\text{real reads} + \text{logical reads} = \sum_{i=1}^{r} (c_i * i * p)$$

The term, r, is the maximum length of candidates.
The term, c_i, is the number of i-candidates.
The term, p, is the number of pages per granule.

For example, data 4 has 11696243 total reads, 35662 real reads plus 11660581 logical reads. Each granule requires 13 pages, 400000/(4096*8). So the total number of pages is equal to 13 * (1654 * 1 + 446298 * 2 + 1807 * 3 + 10 * 4).

6.0 Summary

The use of granules improves the method to find association rules. The AND, SHIFTS, and ADD operations among granules are natural instructions for the general machine, thus making the operations fast. The disadvantage with granules is the costs of creating the temporary table to hold the granules. However, the gain is that the data is converted to a compact form, thus benefiting the whole process.

7.0 Manual

This program searches for association rules using granules or attribute values. The following is the general commands to run this program.

```
1) MAIN CREATE TABLE T1 ( C1 INTEGER, C2 INTEGER, C3
   INTEGER, C4 INTEGER )
2) MAIN GENDATA T1 ROWS 100000
3) MAIN LOAD FROM T1
4) MAIN DISCOVER APRIORI ASSOCIATION FROM T1 SUPPORT 2000
```

Basically, step 1 defines a relation. The schema of the relation is stored in a schema file with the relation name and '.dom' extension. Step 2 procures the data set. The data set is an ASCII file containing numerical values delimited by commas. There is one tuple per row in the file. The numerical values are randomly generated. Furthermore, the values for each attribute have weights, so some will appear more often than other values. The ASCII file has the relation name and '.asc' extension. Step 3 loads the ASCII file into the relation. The ASCII file is read, and the data stored into a data file with the relation name and '.tbl' extension. The data file stores in data pages increments, and each data page holds as many tuples in the relation it can. Step 4 finds associations rules from the relation.

Below is the general syntax for this program.

```
MainCommand := MAIN Commands
TableName   := identifier
Number      := integer
AttributeName := identifier
AttributeNamePair := AttributeName INTEGER
```

```
Method       := GRANULE | APRIORI | APRIORITID |
                APRIORIHYBRID

Commands     := CreateCmd | GenDataCmd | LoadCmd |
                DiscoverCmd

CreateCmd    := CREATE TABLE TableName ( AttributeNamePair+ )
GenDataCmd   := GENDATA TableName [ROWS Number | POPULAR
                Number | COLDOMAINMAX Number ]*
LoadCmd      := LOAD FROM tablename [FORMAT  DBF | ASCII]*
DiscoverCmd := DISCOVER Method ASSOCIATION FROM TableName
                [MAXCOL Number | SUPPORT Number | USESUBSET]*
```

Discover methods:

1) The method, Granule, searches for association rules using granules. The file, tablename.tbl is read and a new file, tablename.tmp is created. The temporary table, tablename.tmp, holds the granules.
2) The method, Apriori, searches for association rules reading through tablename.tbl.
3) The method, AprioriTid, searches for association rules using candidate bar table. The files, tmpfile1 and tmpfile2, holds the candidate bar table. It consists of candidate ids to tuples in the relation.
4) The method, AprioriHybrid, searches for association rules using the method, Apriori, for the first three cycles and the method, AprioriTid, for the other cycles.

Discover options:

1) The option, SUPPORT, specifies the number of tuples required for r-combinations to be declared

association rules. If support is not defined, then the support value is 10% of the number of tuples in the relations.

2) The option, MAXCOL, specifies the longest length of any combinations or association rules. If the maximum column is not specified, then the maximum column is the total number of attribute in the relations.

3) The option, USESUBSET, option specifies that the subset routine is used during the Apriori or during the Apriori portion of AprioriHybrid. If this is not specified, then the subset function is not used. This means the hash-tree table is not created, and every r-candidate is compared to every tuple in the relation.

Works Cited

Agrawal, R., T. Imielinski, and A. Swami, "Mining Association Rules Between Sets of Items in Large Databases," Proceeding of ACM-SIGMOD International Conference on Management of Data, Washington, DC, (May 1993):207-216.

Agrawal, R., R. Srikant, "Fast Algorithms for Mining Association Rules," Proceeding of 20th VLDB Conference, San Tiago, Chile, (1994):487-499.

Chan, Chee-Yong, and Yannis E. Ioannidis, "Bitmap Index Design and Evaluation," Proceeding of ACM-SIGMOD International Conference on Management of Data, Seattle, Washington, (1998):355-366.

Lin, Tsau Young, "Data Mining and Machine Oriented Modeling: A Granular Computing Approach," Journal of Applied Intelligence, Kluwer, Volume 13, Number 2, (September 2000):113-124.

Louie, Eric, and Tsau Young Lin, "Finding Association Rules Using Fast Bit Computation: Machine-Oriented Modeling," Proceedings of the Twelfth International Symposium on Methodologies for Intelligent Systems, Charlotte, NC, USA, (October 2000):486-494.

Ziarko, Wojciech, "Variable Precision Rough Set Model," Journal of Computer and System Sciences, Volume 46, Number 1, (February 1993):39-59.